Tutorial-5

Deepika Negi
CST SPL-1
Roll No: 43

Q1

BFS	DFS
• Uses queue data structure.	• Uses stack data structure.
• Stands for breadth first search.	• Stands for depth first search.
• Can be used to find single source shortest path in a unweighted graph & we reach a vertex with min no. of edges from a source vertex.	• We might traverse through more edges to reach a destination vertex from a source.
• Siblings are visited before the children.	• Children are visited before the siblings.
Applications :	**Applications :**
• Shortest path of min spanning tree for un-weighted graph.	• Detecting cycle in a graph.
• peer to peer networks.	• Path finding.
• Social network weights.	• Topological sorting.
• GPS navigation system.	• Solving puzzles with only one solution.

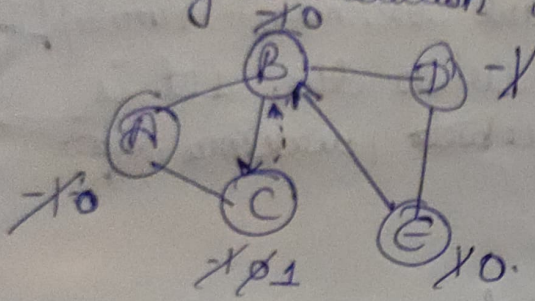
Q2 : In BFS we use queue data structure as queue is used when things don't have to be processed immediately, but have to be processed in FIFO order like BFS.

In DFS stack is used. As DFS uses backtracking, for DFS, we retrieve it from root to the furthest node as much as possible, this is the same idea as LIFO used by stack.

Q3. Dense graph is a graph in which the no. of edges is close to the maximal no. of edges. Sparse graph is a graph in which the no. of edges is close to the minimal no. of edges. It can be disconnected graph.

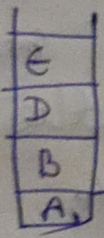
- Adjacency list are preferred for sparse graph & adjacency matrix for dense graph.

Q4. Cycle detection in directed graph (BFS).



-1 = unvisited
0 = visited & in stack.
1 = visited & popped out from stack.

Stack:



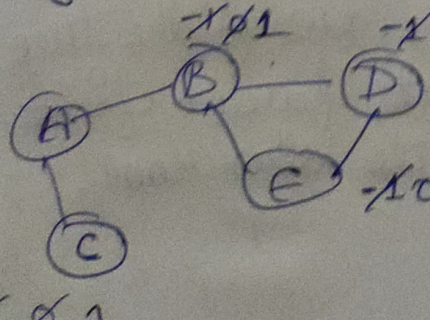
Visited set:
A B D E
B → D → E → B

Parent Map

Vertex	Parent
A	—
B	A
C	B
D	B
E	D

- Here E finds B (adjacency vertex of E) with 0.
⇒ it contains a cycle.

Cycle detection undirected graph



-1 = unvisited
0 = into the queue (node)
1 = traversed.

Queue : | A | B | C | D | E |

Visited set : | A | B | C | D |

when D checks it adjacent vertex with flag 0, the it contain cycle.

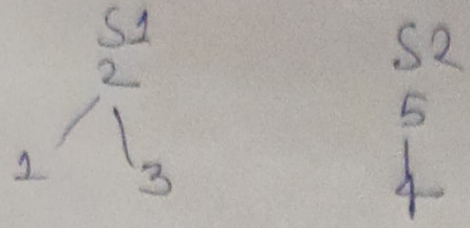
Q5 The disjoint set data structure is also known as union finds data structure & merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets, various operation can be performed on it.

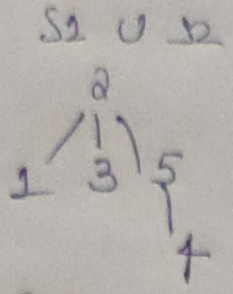
In this case, we can add new sets, we can merge the sets & we can also find the representation member of set. It also allows to find out whether the two elements are in the same set or not efficiently.

Operations on disjoints sets are:

1) Union:

- If S1 & S2 are two disjoints sets, then union S1 US2 is a set of all elements x such that x is in either S1 or S2.
- As the set should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exist.
- Union is achieved by simply making one of the tree as a sub-tree of other i.e. to set parent field of one of the roots of the tree to other root.



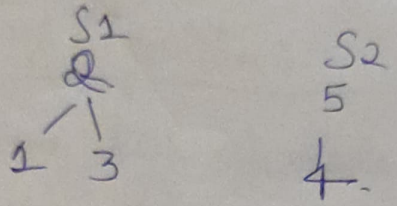


Merge the sets containing x & containing y into one.

(2.) Find

⇒ Given an element x, to find the set containing it.

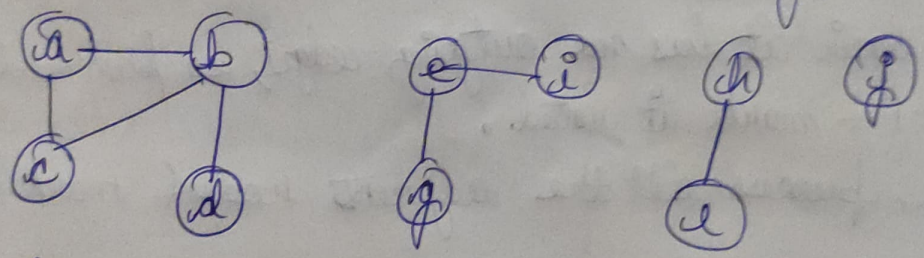
Ex:-



find(3) ⇒ S1 return in which set x belongs.
find(5) ⇒ S2.

(3.) Make-set(x): create a set containing x.

Q7



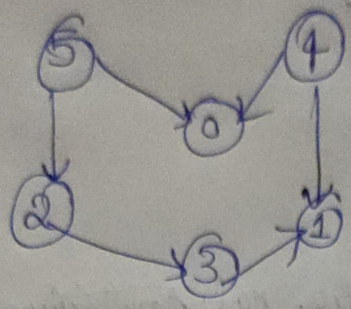
V = {a, b, c, d, e, g, h, i, f, l}

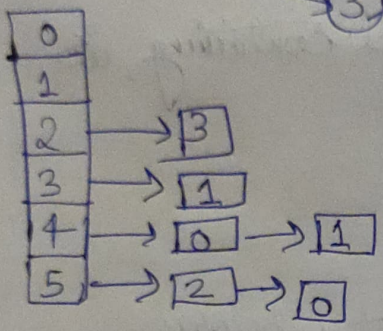
E = {(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,l), (f)}

	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{f}	{l}
(a,b)	{a,b}	{c}	{d}	{e}	{g}	{h}	{i}	{f}	{l}		
(a,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{f}	{l}			
(b,c)	{a,b,c}	{d}	{e}	{g}	{h}	{i}	{f}	{l}			
(b,d)	{a,b,c,d}	{e}	{g}	{h}	{i}	{f}	{l}				
(e,i)	{a,b,c,d}	{e,i}	{g}	{h}	{f}	{l}					
(e,g)	{a,b,c,d}	{e,i,g}	{h}	{f}	{l}						
(h,l)	{a,b,c,d}	{e,i,g}	{h,l}	{f}							
(f)	{a,b,c,d}	{e,i,g}	{h,l}	{f}							

We have
{a, b, c, d}
{e, i, g}
{h, d}
{y}

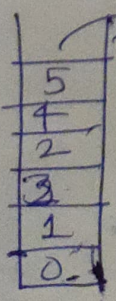
Q8.





Algorithm:-

- Go to node 0, it has no outgoing edges so push node 0 into the stack & mark it visited.
- Goto node 1, again it has no outgoing edges, so push node 1 into the stack & mark it visited.
- Goto node 2, process all the adjacent nodes & mark node 2 visited.
- Node 3 is already visited so continue with next node.
- Go to node 4, all its adjacent nodes are already visited so push node 4, into the stack & mark it visited.
- Goto node 5, all its adjacent nodes are already visited, so push node 5 into the stack & mark it visited.



5 4 2 3 1 0 output

09. Heap is generally preferred for priority queue implementation because heaps provide better performance compared to arrays or linked list.

Algorithms where priority queue is used:

1. Dijkstra's :- (Shortest path algorithm).

→ where the graph is stored in the form of adjacently list or matrix, priority queue can be used to extract min. efficiently when implementing dijkstra's algorithm.

2. Prim's Algorithm :-

→ To store key of nodes and extract min. key node at every step.

Min heap :-	Max heap :-
• For every pair of the parent & descendant child node, the parent node always has lower value than descended child node.	• For every pair of parent & descendant child node, the parent node has greater value than descended child node.
• The value of nodes inc. as we traverse from root to leaf node.	• The value of node dec. as we traverse from root to leaf node.
• Root node has lowest value.	• Root node has greatest value.